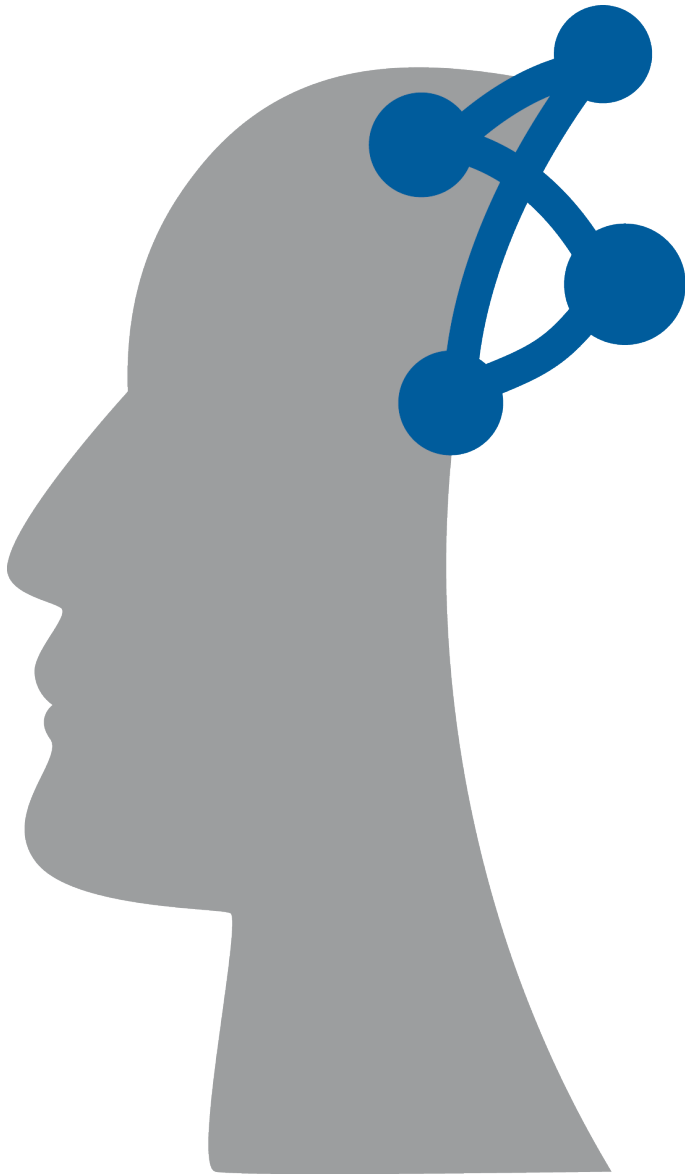
Helmholtz Portfolio Theme

Supercomputing and Modeling for the Human Brain



WP 4: Supporting Software

T 4.6: Methods & Infrastructure for software development

21 March 2014 | Abigail Morrison
Alexander Peyser
Jochen Eppler

Methods & Infrastructure for software development

Helmholtz Portfolio Theme SMHB Supercomputing and Modeling for the Human Brain

M18

Reduce complexity and increase maintainability with Cython

M24

Prototype software documentation site

Continuous Integration as a Service for Neuroinformatics

M36

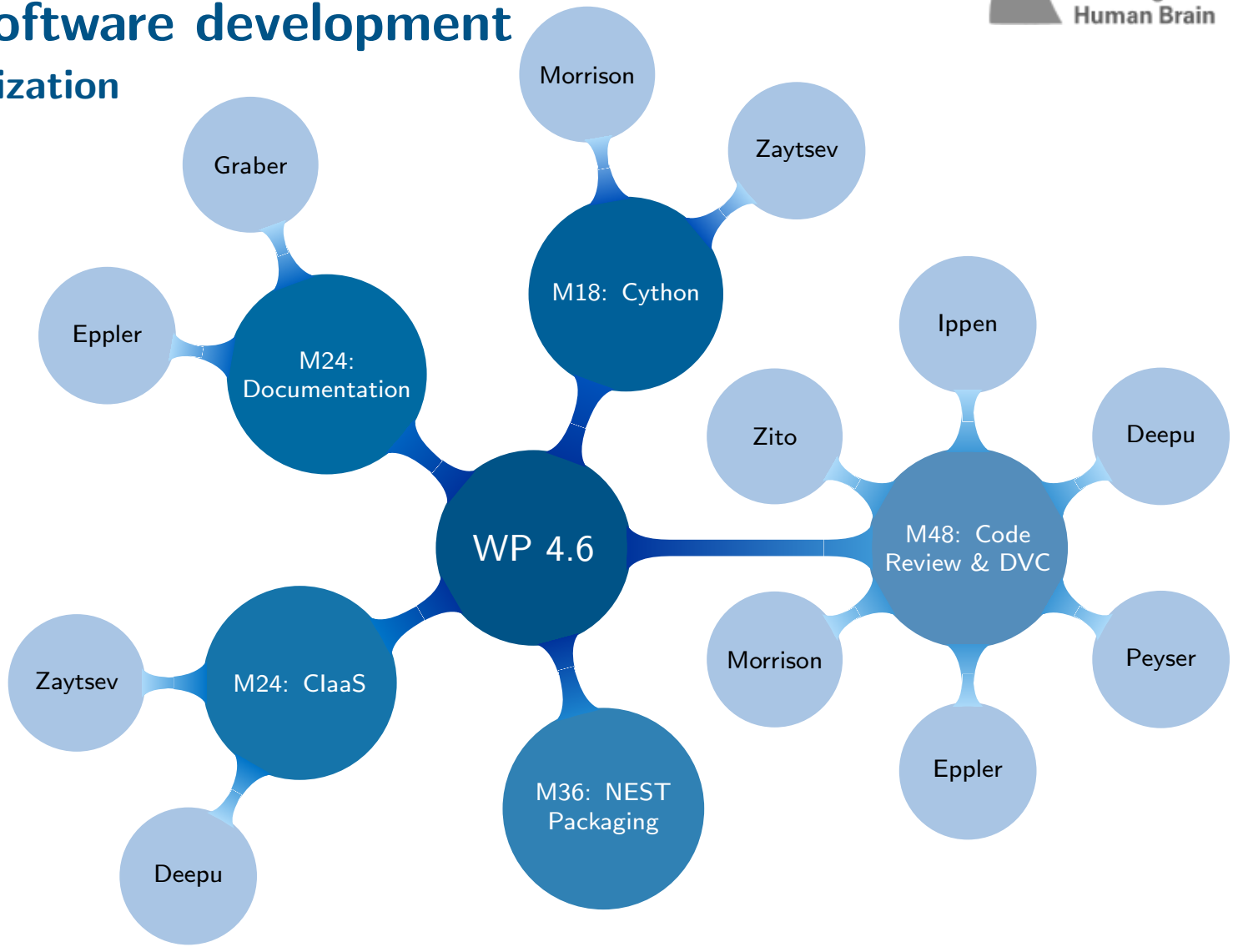
NEST packaging for critical distributions

M48

Prototype of code review platform & distributed version control

Methods & Infrastructure for software development

Organization



M18: Cython

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

Milestone

Completed study on the use of Cython to reduce complexity and error likelihood and increase maintainability of interfaces to neural network simulators (see also T3.4 milestone 2)

Status

PyNEST bindings for NEST (Neural Simulation Tool) rewritten in Cython supporting both Python 2 & 3

Paper written documenting maintainability and performance issues [Zaytsev and Morrison, 2014]

M18: Cython

frontiers in
NEUROINFORMATICS

ORIGINAL RESEARCH ARTICLE
published: 14 March 2014
doi: 10.3389/fninf.2014.00023

CyNEST: a maintainable Cython-based interface for the NEST simulator

***Yury V. Zaytsev*[1,2]* and *Abigail Morrison*[1,3,4]**

[1] *Simulation Laboratory Neuroscience – Bernstein Facility for Simulation and Database Technology, Institute for Advanced Simulation, Jülich Aachen Research Alliance, Jülich Research Center, Jülich, Germany*
[2] *Faculty of Biology, Albert-Ludwig University of Freiburg, Freiburg im Breisgau, Germany*
[3] *Institute for Advanced Simulation (IAS-6), Theoretical Neuroscience and Institute of Neuroscience and Medicine (INM-6), Computational and Systems Neuroscience, Jülich Research Center and JARA, Jülich, Germany*
[4] *Institute of Cognitive Neuroscience, Faculty of Psychology, Ruhr-University Bochum, Bochum, Germany*

Edited by:
Yaroslav O. Halchenko, Dartmouth College, USA

Reviewed by:
Mikael Djurfeldt, KTH Royal Institute of Technology, Sweden
Laurent U. Perrinet, Centre National de la Recherche Scientif que, France

***Correspondence:**
Yury V. Zaytsev, Simulation Laboratory Neuroscience - Bernstein Facility for Simulation and Database Technology, Institute for Advanced Simulation, Jülich Aachen Research Alliance, Jülich Research Center, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany
e-mail: zaytsev@fz-juelich.de

NEST is a simulator for large-scale networks of spiking point neuron models (Gewaltig and Diesmann, 2007). Originally, simulations were controlled via the Simulation Language Interpreter (SLI), a built-in scripting facility implementing a language derived from PostScript (Adobe Systems, Inc., 1999). The introduction of PyNEST (Eppler et al., 2008), the Python interface for NEST, enabled users to control simulations using Python. As the majority of NEST users found PyNEST easier to use and to combine with other applications, it immediately displaced SLI as the default NEST interface. However, developing and maintaining PyNEST has become increasingly diff cult over time. This is partly because adding new features requires writing low-level C++ code intermixed with calls to the Python/C API, which is unrewarding. Moreover, the Python/C API evolves with each new version of Python, which results in a proliferation of version-dependent code branches. In this contribution we present the re-implementation of PyNEST in the Cython language, a superset of Python that additionally supports the declaration of C/C++ types for variables and class attributes, and provides a convenient foreign function interface (FFI) for invoking C/C++ routines (Behnel et al., 2011). Code generation via Cython allows the production of smaller and more maintainable bindings, including increased compatibility with all supported Python releases without additional burden for NEST developers. Furthermore, this novel approach opens up the possibility to support alternative implementations of the Python language at no cost given a functional Cython back-end for the corresponding implementation, and also enables cross-compilation of Python bindings for embedded systems and supercomputers alike.

Keywords: Python language, neural simulator, maintainability, technical debt, HPC

M18: Cython

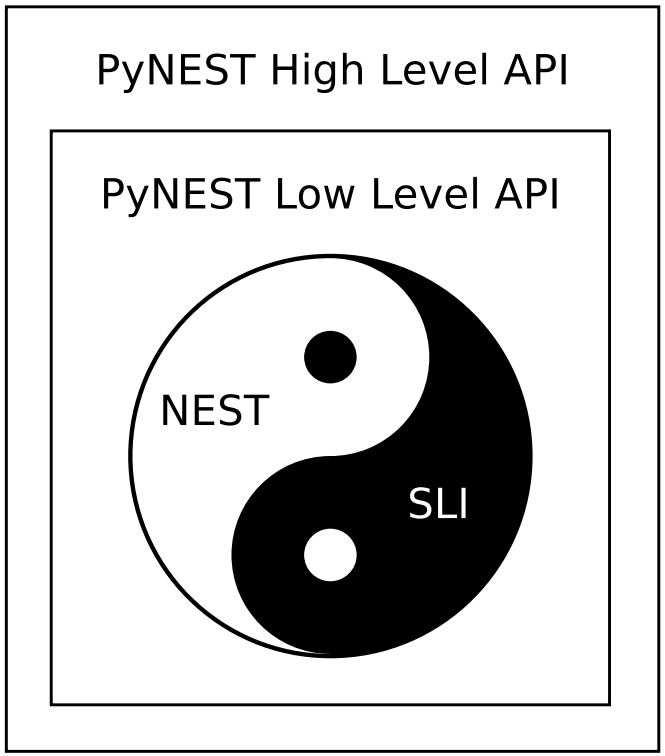


FIGURE 1 | Diagram depicting the design of PyNEST, the Python bindings for the NEST simulator.

Table 3 | The runtime and memory consumption measurements performed on the untraced version of the simplif ed Hill-Tononi model using single-threaded simulation on the hardware/software setup described in the main text.

	Median	Minimum	Maximum
RUNTIME OF THE MODEL [s]			
PyNEST	64.8	63.8	66.9
CyNEST	66.1	65.5	71.6
PEAK MEMORY USAGE [MiB]			
PyNEST	327	327	328
CyNEST	329	329	329

The runtime (which includes the collection of the results and the production of the graphics) was obtained using the shell TIME *command and the memory usage was recorded using Massif. Each experiment was repeated n = 15 times, alternating the measurements for PyNEST and CyNEST.*

M24: Software documentation site

Example documentation, cont

Milestone

Prototype implementation of a software documentation site

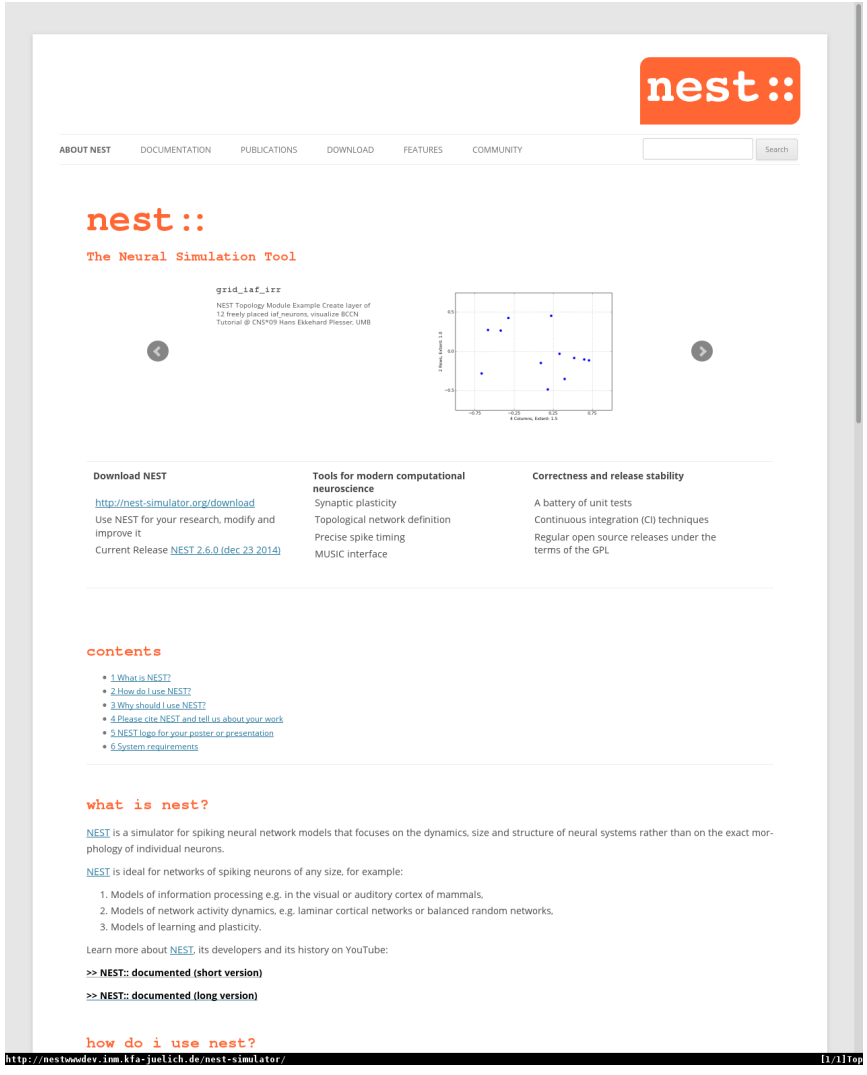
Status

Automated documentation engine developed by Steffen Graber using Sphinx + other tools to scrape code to produce website

Website prototyped as `http://nestwwwdev.inm.kfa-juelich.de/nest-simulator`

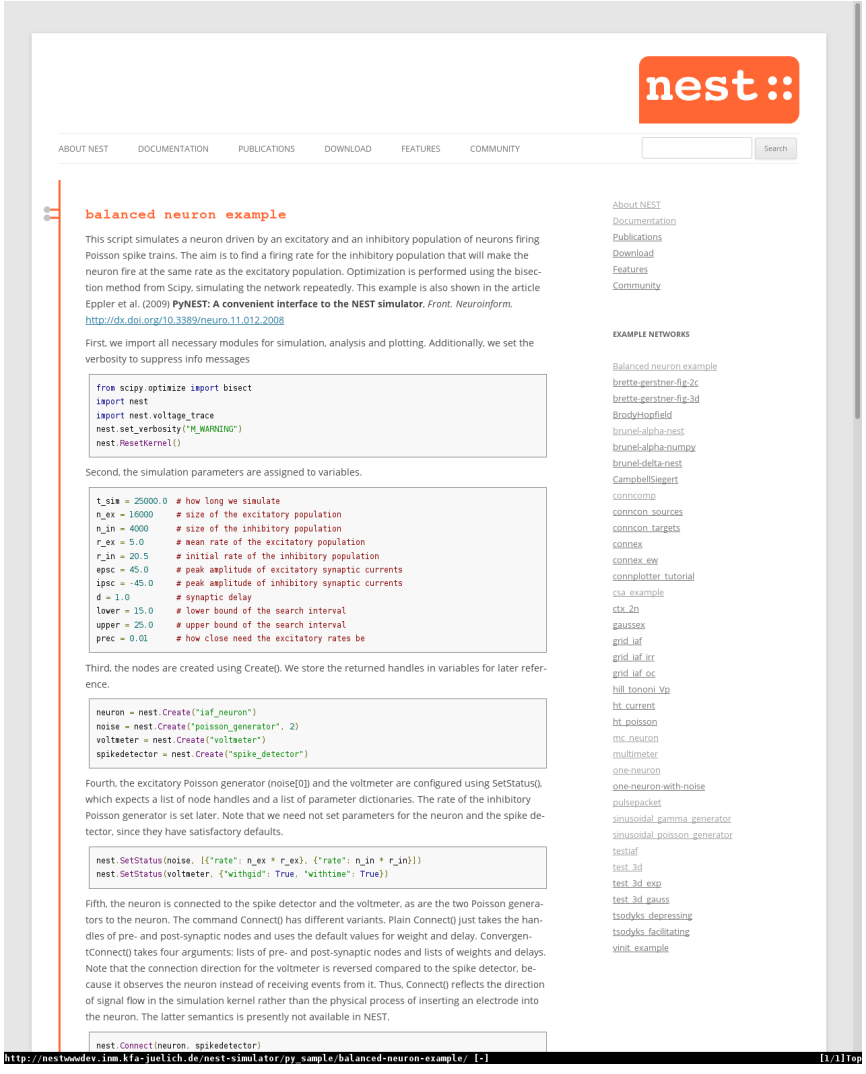
M24: Software documentation site

Example documentation, cont



M24: Software documentation site

Example documentation, cont



nest ::

ABOUT NEST | DOCUMENTATION | PUBLICATIONS | DOWNLOAD | FEATURES | COMMUNITY | Search

balanced neuron example

This script simulates a neuron driven by an excitatory and an inhibitory population of neurons firing Poisson spike trains. The aim is to find a firing rate for the inhibitory population that will make the neuron fire at the same rate as the excitatory population. Optimization is performed using the bisection method from Scipy, simulating the network repeatedly. This example is also shown in the article Eppler et al. (2009) **PyNEST: A convenient interface to the NEST simulator**. *Front. Neuroinform.* http://dx.doi.org/10.3389/neuro.11.012.2008

First, we import all necessary modules for simulation, analysis and plotting. Additionally, we set the verbosity to suppress info messages

```
from scipy.optimize import bisect
import nest
import nest.voltage_trace
nest.set_verbosity("M_WARNING")
nest.ResetKernel()
```

Second, the simulation parameters are assigned to variables.

```
t_sim = 25000.0  # how long we simulate
n_ex = 16000     # size of the excitatory population
n_in = 4000      # size of the inhibitory population
r_ex = 5.0       # mean rate of the excitatory population
r_in = 20.5      # initial rate of the inhibitory population
epsc = 45.0      # peak amplitude of excitatory synaptic currents
ipsc = -45.0     # peak amplitude of inhibitory synaptic currents
d = 1.0          # synaptic delay
lower = 15.0     # lower bound of the search interval
upper = 25.0     # upper bound of the search interval
prec = 0.01      # how close need the excitatory rates be
```

Third, the nodes are created using Create(). We store the returned handles in variables for later reference.

```
neuron = nest.Create("iaf_neuron")
noise = nest.Create("poisson_generator", 2)
voltmeter = nest.Create("voltmeter")
spikedetector = nest.Create("spike_detector")
```

Fourth, the excitatory Poisson generator (noise[0]) and the voltmeter are configured using SetStatus(), which expects a list of node handles and a list of parameter dictionaries. The rate of the inhibitory Poisson generator is set later. Note that we need not set parameters for the neuron and the spike detector, since they have satisfactory defaults.

```
nest.SetStatus(noise, [{"rate": n_ex * r_ex}, {"rate": n_in * r_in}])
nest.SetStatus(voltmeter, {"withgid": True, "withtime": True})
```

Fifth, the neuron is connected to the spike detector and the voltmeter, as are the two Poisson generators to the neuron. The command Connect() has different variants. Plain Connect() just takes the handles of pre- and post-synaptic nodes and uses the default values for weight and delay. ConvergentConnect() takes four arguments: lists of pre- and post-synaptic nodes and lists of weights and delays. Note that the connection direction for the voltmeter is reversed compared to the spike detector, because it observes the neuron instead of receiving events from it. Thus, Connect() reflects the direction of signal flow in the simulation kernel rather than the physical process of inserting an electrode into the neuron. The latter semantics is presently not available in NEST.

```
nest.Connect(neuron, spikedetector)
```

About NEST
Documentation
Publications
Download
Features
Community

EXAMPLE NETWORKS

Balanced neuron example
brette-gerstner-fig-2c
brette-gerstner-fig-3d
BrodyHopfield
brunel-alpha-nest
brunel-alpha-numpy
brunel-delta-nest
CampbellSiegert
conncomp
conncon_sources
conncon_targets
connex
connex_ew
connplotter_tutorial
csa_example
ctx_2n
gaussex
grid_iaf
grid_iaf_irr
grid_iaf_oc
hill_tononi_Vp
ht_current
ht_poisson
mc_neuron
multimeter
one-neuron
one-neuron-with-noise
pulsepacket
sinusoidal_gamma_generator
sinusoidal_poisson_generator
testiaf
test_3d
test_3d_exp
test_3d_gauss
tsodyks_depressing
tsodyks_facilitating
vinit_example

http://nestwwwdev.inm.kfa-juelich.de/nest-simulator/py_sample/balanced-neuron-example/ [-] [1/1]Top

M24: Software documentation site

Example documentation, cont

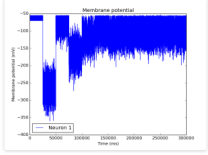
```
def output_rate(guess):
    print("Inhibitory rate estimate: %5.2f Hz" % guess)
    rate = float(abs(n_in * guess))
    nest.SetStatus([noise[1]], "rate", rate)
    nest.SetStatus(spikedetector, "n_events", 0)
    nest.Simulate(t_sim)
    out = nest.GetStatus(spikedetector, "n_events")[0] * 1000.0 / t_sim
    print("  -> Neuron rate: %6.2f Hz (goal: %4.2f Hz)" % (out, r_ex))
    return out
```

The function takes the firing rate of the inhibitory neurons as an argument. It scales the rate with the size of the inhibitory population and configures the inhibitory Poisson generator (noise[1]) accordingly. Then, the spike-counter of the spike detector is reset to zero. The network is simulated using Simulate(), which takes the desired simulation time in milliseconds and advances the network state by this amount of time. During simulation, the spike detector counts the spikes of the target neuron and the total number is read out at the end of the simulation period. The return value of output_rate() is the firing rate of the target neuron in Hz. Second, the scipy function bisect() is used to determine the optimal firing rate of the neurons of the inhibitory population.

```
in_rate = bisect(lambda x: output_rate(x) - r_ex, lower, upper, xtol=prec)
print("Optimal rate for the inhibitory population: %.2f Hz" % in_rate)
```

The function bisect() takes four arguments: first a function whose zero crossing is to be determined. Here, the firing rate of the target neuron should equal the firing rate of the neurons of the excitatory population. Thus we define an anonymous function (using lambda) that returns the difference between the actual rate of the target neuron and the rate of the excitatory Poisson generator, given a rate for the inhibitory neurons. The next two arguments are the lower and upper bound of the interval in which to search for the zero crossing. The fourth argument of bisect() is the desired relative precision of the zero crossing. Finally, we plot the target neuron's membrane potential as a function of time.

```
nest.voltage_trace.from_device(voltmeter)
```



Download python file: balancedneuron.py
Download ipython notebook file: balancedneuron.py.ipynb

```
# -*- coding: utf-8 -*-
#
# balancedneuron.py
#
# This file is part of NEST.
#
# Copyright (C) 2004 The NEST Initiative
#
# NEST is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 2 of the License, or
# (at your option) any later version.
#
# NEST is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with NEST.  If not, see <http://www.gnu.org/licenses/>.
```

http://nestwwwdev.inm.kfa-juelich.de/nest-simulator/py_sample/balanced-neuron-example/ [-] [1/1]78%

M24: Continuous Integration as a Service

Automated Jenkins builds

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

Milestone

Continuous Integration as a Service (ClaaS) available for neuroinformatics applications

Status

ClaaS has been implemented for

NEST	Neural Simulation Tool
PyNN	Simulator-independent language for building neuronal network models
MUSIC	C++ library for exchanging data during runtime for large scale neuronal network simulators
Topographica	Computational modeling of neural maps

M24: Continuous Integration as a Service

Automated Jenkins builds

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

Milestone

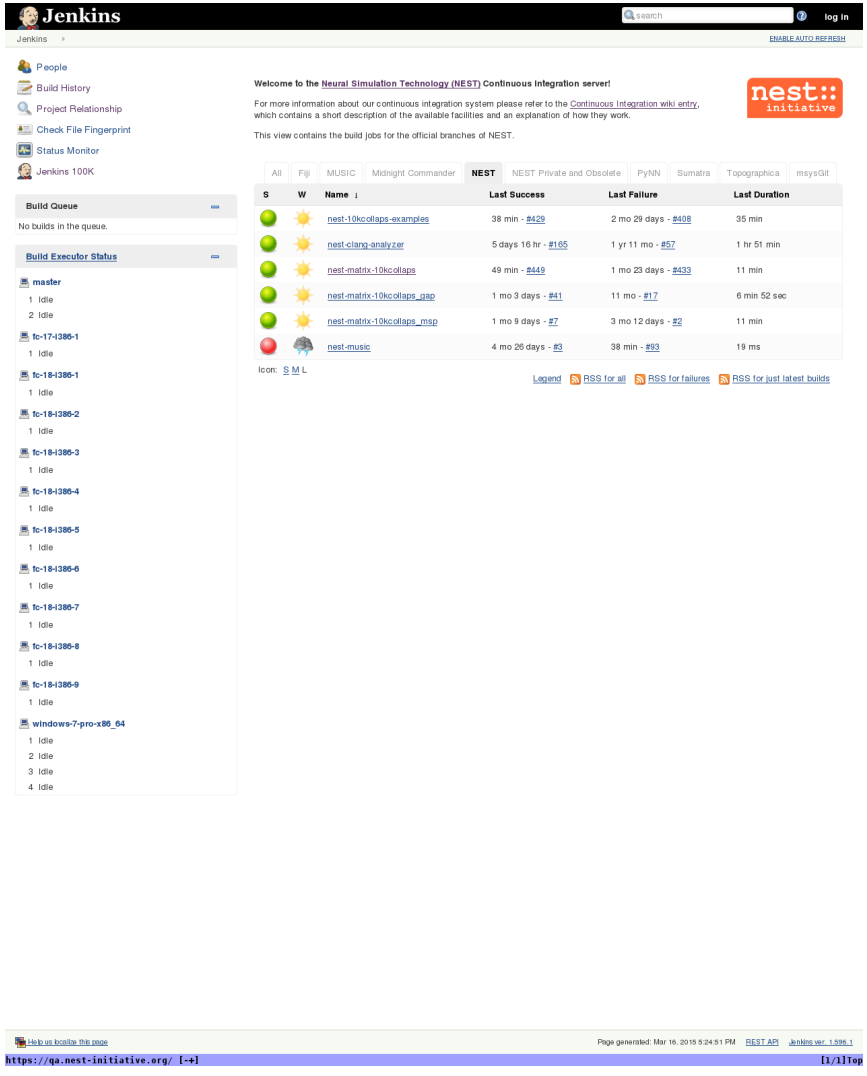
Continuous Integration as a Service (ClaaS) available for neuroinformatics applications

Status

Infrastructure	is available at `https://qa.nest-initiative.org`
Developers	Yury Zaytsev, Lekshmi Deepu

M24: Continuous Integration as a Service

Automated Jenkins builds



M36: NEST packaging

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

Milestone

NEST packaged and queued for inclusion in critical distributions

Status

Exchanged with M48:
Prototype of code review platform & distributed version control

---

This exchange prioritizes developer access and quality control

M48: Code review & DVC platform

Leverage public infrastructure

Helmholtz Portfolio Theme
SMHB
Supercomputing and Modeling for the Human Brain

Milestone

Prototype implementation of a code review based development platform integrated with distributed version control

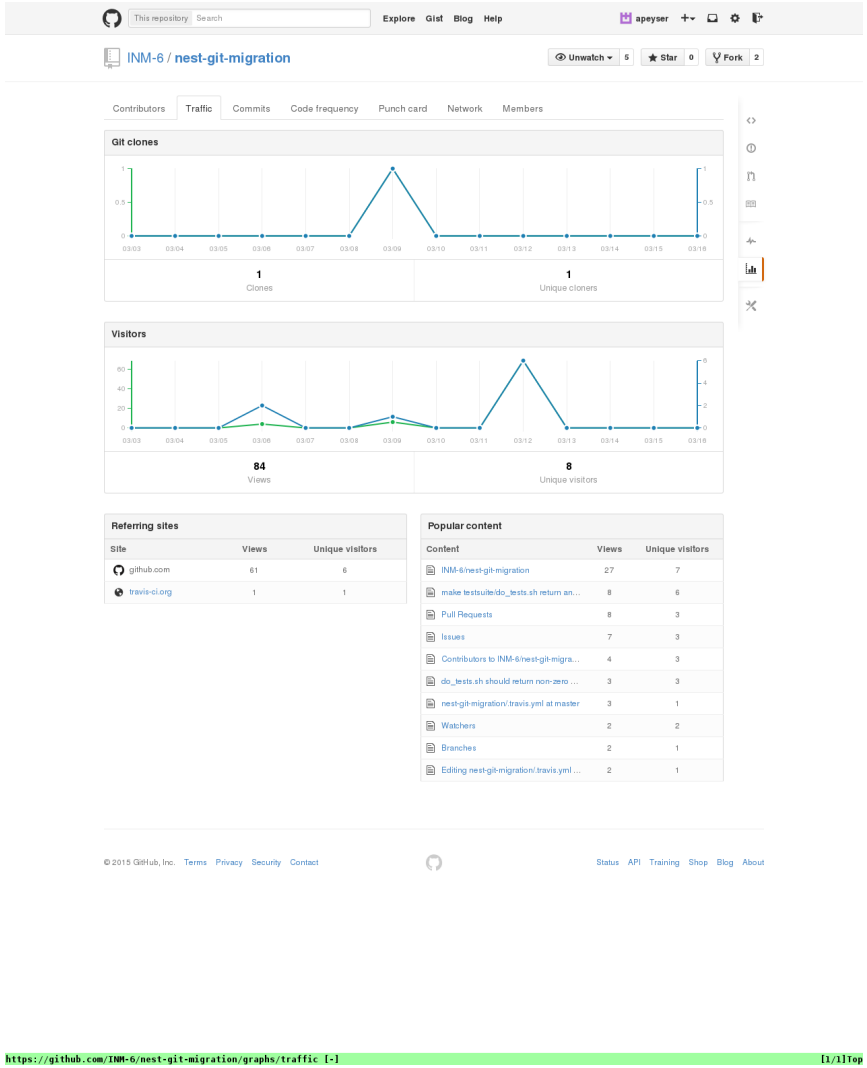
Status

Solution: We do as Google does, and use GitHub

Prototype	site is being worked on as INM-6/nest-git-migration
Code Review	process is being developed using GitHub tools
Cont Int	is through a GitHub–Travis bridge
Documentation	of use cases and processes are being developed to facilitate reuse

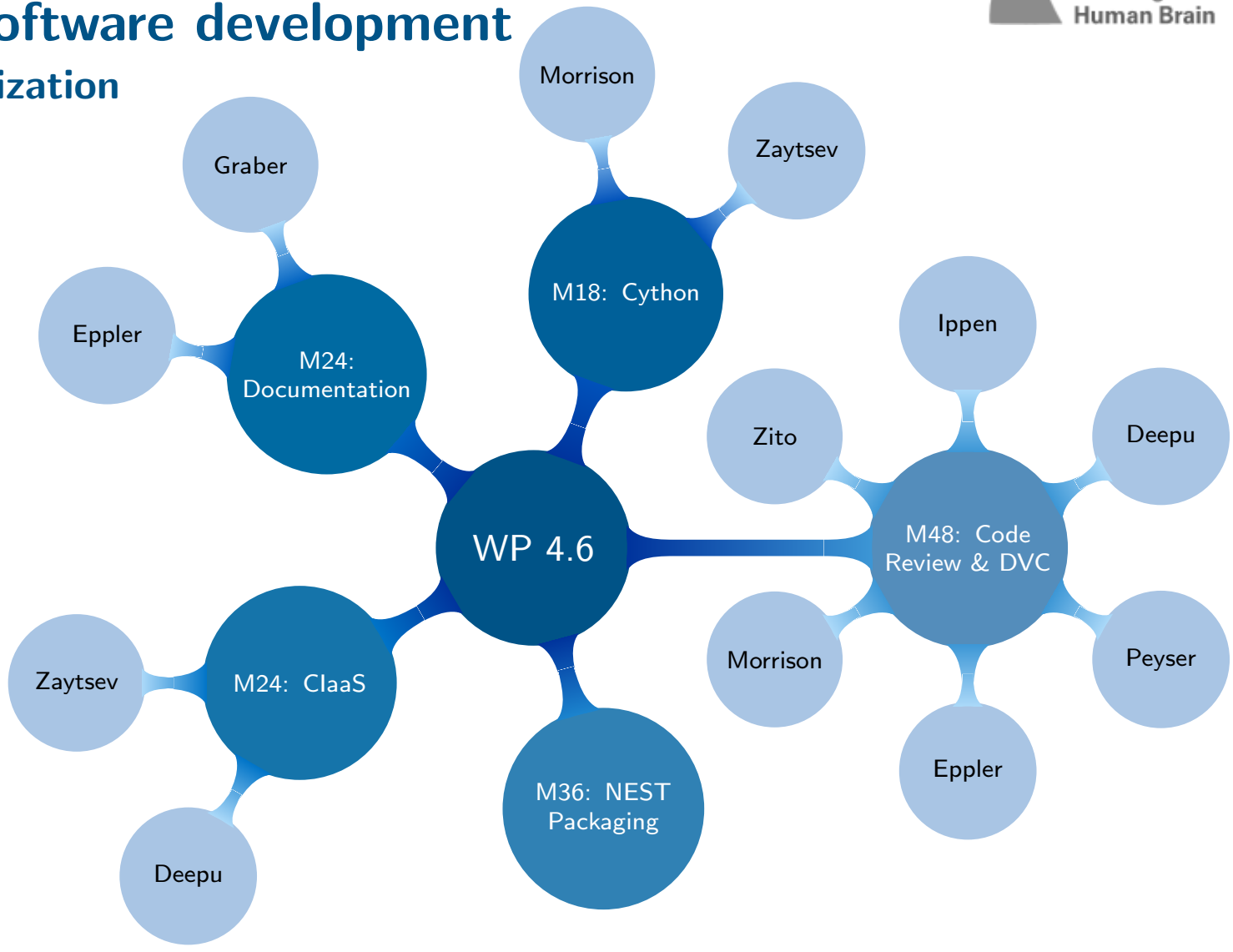
M48: Code review & DVC platform

Leverage public infrastructure



Methods & Infrastructure for software development

Organization



References

Yury V. Zaytsev and Abigail Morrison.
CyNEST: a maintainable Cython-based interface for the NEST simulator.
Frontiers in Neuroinformatics, 8(23), 2014.
doi: 10.3389/fninf.2014.00023.